Estimation of Round-off Errors in OpenMP Codes

Pacôme Eberhart, Julien Brajard, Pierre Fortin, Fabienne Jézéquel

Estimation of round-off errors in OpenMP codes

P. Eberhart[1], J. Brajard[2,3], P. Fortin[1], and F. Jézéquel[1,4]

[1] Sorbonne Universités, UPMC Univ Paris 06, CNRS, UMR 7606, LIP6, F-75005, Paris, France
[2] Sorbonne Universités, UPMC Univ Paris 06, CNRS-IRD-MNHN, LOCEAN, F-75005, Paris, France
[3] Inria Paris, 2 rue Simone Iff, Paris, France
[4] Université Panthéon-Assas, 12 place du Panthéon, 75231 Paris CEDEX 05, France
{Pacome.Eberhart,Julien.Brajard,Pierre.Fortin,Fabienne.Jezequel}@upmc.fr

Abstract. It is crucial to control round-off error propagation in numerical simulations, because they can significantly affect computed results, especially in parallel codes like OpenMP ones. In this paper, we present a new version of the CADNA library that enables the numerical validation of OpenMP codes. With a reasonable cost in terms of execution time, it enables one to estimate which digits in computed results are affected by round-off errors and to detect numerical instabilities that may occur during the execution. The interest of this new OpenMP-enabled CADNA version is shown on various applications, along with performance results on multi-core and many-core (Intel Xeon Phi) architectures.

Keywords: CADNA library, Discrete Stochastic Arithmetic, floating-point arithmetic, numerical validation, multi-core architectures, many-core architectures, OpenMP, round-off error

1 Introduction

The power of computational resources tends to increase and so does the number of floating-point operations performed in numerical simulations. Unfortunately each floating-point operation may generate a round-off error. These errors can accumulate and significantly affect the computed results. In parallel applications, more computations can lead to more round-off errors, and the order of computations can change with respect to the sequential execution or to another parallel execution: this can cause different round-off error propagations and hence different numerical results

The CADNA library[5] [4, 8] that implements DSA (Discrete Stochastic Arithmetic) [16, 17] enables one to estimate round-off errors in numerical simulation codes. In this paper we present a CADNA version for the numerical validation of codes using the OpenMP standard which is widely used for parallel numerical simulations. We describe how this new CADNA version has been developed taking into account constraints related to both OpenMP and DSA. We show the

[5] URL address: http://cadna.lip6.fr

benefit of CADNA and analyse its cost in terms of execution time for various OpenMP programs: several benchmark codes and also a real-life application.

As far as related work is concerned, DSA is sometimes compared to interval arithmetic [1, 10] that takes into accout round-off errors and computes, instead of each floating-point result, an interval with guaranteed bounds. However, with a naive implementation of interval arithmetic in a code, these bounds may overestimate the effective error. Therefore the use of interval arithmetic in an application requires specific algorithms and methods. Interval versions of linear algebra kernels have been proposed for multicore architectures [5, 11, 14, 19] and have been implemented using OpenMP pragmas or a standard POSIX threads [6] library.

The rest of the paper is organized as follows. Section 2 presents the CADNA library and its use for the numerical validation of sequential codes. Section 3 describes the new OpenMP-enabled CADNA version developed to control the numerical quality of OpenMP codes. Section 4 presents, for various OpenMP applications, the interest of this CADNA version, as well as some performance and numerical results. Finally concluding remarks are given in Sect. 5.

2 The CADNA library

2.1 Principles of DSA (Discrete Stochastic Arithmetic)

The CADNA (Control of Accuracy and Debugging for Numerical Applications) library [4, 8] that implements DSA (Discrete Stochastic Arithmetic) [16, 17] enables one to estimate round-off errors in any scientific code written in C, C++ or Fortran. CADNA uses a random rounding mode: at each elementary operation, the result is rounded up or down with the probability 0.5. The computer's deterministic arithmetic, therefore, is replaced by a stochastic arithmetic where each arithmetic operation is performed N times before the next one is executed, thereby propagating the round-off error differently each time. For each computed result, we obtain N samples $R_1, \ldots, R_N$. The value of the computed result $\overline{R}$ is chosen to be the mean value of $\{R_i\}$ and, if no overflow occurs, the number of exact significant digits in $\overline{R}$, $C_{\overline{R}}$, can be estimated as

$$C_{\overline{R}} = \log_{10}\left(\frac{\sqrt{N}\,|\overline{R}|}{\sigma\tau_\beta}\right) \text{ with } \overline{R} = \frac{1}{N}\sum_{i=1}^{N} R_i \text{ and } \sigma^2 = \frac{1}{N-1}\sum_{i=1}^{N}\left(R_i - \overline{R}\right)^2.$$

τ_β is the value of Student's distribution for $N-1$ degrees of freedom and a probability level $1-\beta$. In practice $\beta = 0.05$ and $N = 3$. Thus we can get a 95% confidence interval on the number of exact significant digits of the computed result. DSA is based on an hypothesis on the distribution of round-off errors. It has been shown that in case this hypothesis is not rigourously satisfied the accuracy estimation is not altered if it is considered as exact up to one digit. The complete theory can be found in [16].

The validity of $C_{\overline{R}}$ is compromised if the two operands in a multiplication or the divisor in a division are not significant. Therefore CADNA performs a so-called *self-validation* that consists in controlling all multiplications and divisions

during the execution of the code. This self-validation has led to the parallel computation of the N samples R_i and also to the concept of computational zero [15] defined as either a mathematical zero or a number without any significance, i.e. numerical noise. From this concept, relational operators that take into accout round-off error propagation have been defined [16].

2.2 Numerical validation of sequential codes using CADNA

The CADNA library allows one to use new numerical types: the stochastic types. In practice, classic floating-point variables are replaced by the corresponding stochastic variables, which are composed of three floating-point values and one integer to store the accuracy. The library contains the definition of all arithmetic operations and order relations for the stochastic types. Only the exact significant digits of a stochastic variable are printed or "@.0" if it is numerical noise. Because all operators are redefined for stochastic variables, the use of CADNA in a program requires only a few modifications: essentially changes in the declarations of variables and in input/output statements. CADNA has been successfully used for the numerical validation of academic and industrial simulation codes in various domains such as astrophysics, atomic physics, chemistry, climate science, fluid dynamics, geophysics [9].

We rely here on the latest CADNA version [4], which performs better with sequential codes and also enables the numerical validation of vectorized codes. Unlike in the previous CADNA version, the rounding mode is not explicitly changed in this version. It is set to rounding towards $+\infty$ once, in the `cadna_init` initialisation function that must be called at the beginning of the user program. The random rounding mode is then emulated by taking advantage of arithmetic properties of rounding towards $+\infty$.

CADNA can detect numerical instabilities which occur during the execution of the code. These instabilities are of several types.

- Self-validation: both operands in a multiplication, the divisor in a division, or the first argument of the power function are not significant.
- Instability in an intrinsic or a mathematical function: non significant argument in such a function.
- Branching instability: indeterminism in a branching test.
- Cancellation: sudden loss of accuracy on an addition or a subtraction.

At the end of the run, the total number of instabilities and each type of instability together with their occurrences are printed. If instabilities have occurred, their source need to be identified and, if necessary, the code changed. The user can specify the instabilities to be detected. One may choose, for instance, to activate only self-validation, to detect all types of instabilities or to deactivate the detection of instabilities.

3 CADNA for OpenMP codes

We detail here how we have designed an OpenMP-enabled CADNA version for C/C++ codes, although up to our knowledge such a version could also be developed for Fortran codes.

Compatibility check. Our CADNA implementation requires the rounding mode of each thread being set to rounding towards $+\infty$. But to our knowledge, there is no guarantee in the OpenMP standard that the worker threads have the same rounding mode as their master thread (though vendor specific extension may ensure this such as the `KMP_INHERIT_FP_CONTROL` environment variable for the Intel compiler [7]).

We thus have to check the compatibility between each OpenMP implementation used and CADNA. Within the `cadna_init` call (considered as not performed inside an OpenMP parallel region), we hence first set the rounding mode of the master thread to rounding towards $+\infty$, and then check in an OpenMP parallel region that all worker threads have inherited their rounding mode from the master thread. This is performed thanks to a software test: in double precision, only rounding towards $+\infty$ ensures that 1.0 + 1.0e−20 != 1.0

This enables us to check OpenMP implementations that fork new worker threads at each OpenMP parallel region. If the parallel region within the `cadna_init` call is the first one in the program, this test enables us to also check for OpenMP implementations that create worker threads once at the first OpenMP parallel region, and then retrieve worker threads from a thread pool.

However, when partially analyzing a large OpenMP code with CADNA, it is convenient to be able to consider that the `cadna_init` call is performed after some parallel regions. In such case, and considering that all parallel regions (the ones within the `cadna_init` call and the ones in the CADNA instrumented code) have the same number of threads (condition hereafter referred to as (*)), we check thread pool based OpenMP implementations with the following test within the `cadna_init` call: we set to rounding towards $+\infty$ the rounding mode of all threads in a second parallel region, and we check in a third parallel region that the rounding mode of each thread has been correctly saved. If this second check fails, we consider that the OpenMP execution environment is not compatible with CADNA.

In our tests, both GOMP and Intel implementations have found to be compatible with CADNA, whether `cadna_init` is called prior to any OpenMP parallel region or not.

Random rounding mode. The next issue lies in the random number generation. Thanks to the new version of CADNA (see Sect. 2.2) used here, we now rely on a random number generator that was designed to be easily replicated in each execution flow; this was indeed required for a SIMD execution of CADNA where the random generator was replicated for each scalar lane of the SIMD unit. It is thus straightforward to have a distinct random generator in each

OpenMP thread: the three required variables are defined as `threadprivate` and are initialized in a parallel region within the `cadna_init` call.

However, in order to ensure that the values of these threadprivate variables will persist from one parallel region to the next one, we have to check the conditions listed in Sect. 2.14.2 of the OpenMP 4.0 reference [2], which we hereafter denote as (**) and which we summarize here as: same number of threads and same affinity policies for all parallel regions, no nested parallelism and no dynamic adjustment of the number of threads. It can be noticed that the conditions (**) encompass the condition (*).

Instability detection. In an OpenMP application instrumented with CADNA, the counters dedicated to the numerical instabilities (see Sect. 2.2) can be concurrently incremented by multiple threads, which can lead to data race conditions. We have thus protected the updates of these counters with OpenMP `atomic` constructs.

OpenMP reductions. When an OpenMP application includes an OpenMP reduction on floating-point variables, instrumenting such a reduction with CADNA implies to perform the corresponding reduction operation with stochastic variables. This is possible since OpenMP 4.0 thanks to the `declare reduction` construct and the redefinition of all arithmetic operators in the CADNA codes. Currently the `+`, `-` and `*` operators are supported.

Atomic constructs on stochastic variables. If an OpenMP application contains `atomic` constructs, the code cannot be instrumented as is with CADNA. As specified in the OpenMP 4.0 reference [2], an `atomic` statement should involve scalar types, and cannot thus be applied on the CADNA stochastic types.

A CADNA-redefined arithmetic operation implies three floating-point IEEE operations, some bit manipulations (depending on the type of the arithmetic operation and on the rounding mode), and some instability detections (depending on the user specification). During this sequence of operations, we have to ensure that the variables are accessed by only one thread. The instability detection depends indeed on the result of the three floating-point computations. We have therefore chosen to include such CADNA-overloaded operation in a `critical` block. In practice, each `atomic` construct is replaced by a `critical` construct in the user code, and one can use distinct `critical` regions (with distinct names) when accessing distinct variables or arrays.

Provided that the conditions (**) are ensured, CADNA can now be used in OpenMP codes. We emphasize that, except for the `atomic` constructs, the OpenMP-enabled CADNA version does not require additional modifications to the user code other than those required by the sequential CADNA library.

4 Performance tests and application to OpenMP codes

For our tests, we use two different HPC servers. The first one, named *CPU-server*, is composed of two Intel E5-2660 CPUs (each having 8 cores with 2-way SMT at 2.20 GHz) and 32 GB of memory. The second one is a 5110P Xeon Phi (Knight Corner), used in native mode as a distant server and named *Xeon-Phi*, with 60 cores (4-way SMT at 1.053 GHz) and 8 GB of dedicated memory.

In all our tests, the number of threads in each parallel region is determined according to the `OMP_NUM_THREADS` environment variable. Moreover, the same affinity policies are set for all parallel region, and no nested parallelism as well as no dynamic adjustment of the number of threads are used on both servers in order to meet the conditions (**) (see Sect. 3).

Unless otherwise mentioned, we use here CADNA with only self-validation activated, since this is its classic usage. It has to be noticed that all runs are scalar ones (no vectorization used): as shown in [4] it is possible to use CADNA for SIMD codes but this requires to access the SIMD lane identifier. To our knowledge this is currently not possible with OpenMP directives, and we therefore rely on the SPMD-on-SIMD programming paradigm (e.g. with the Intel SPMD Program Compiler - `ispc`[6]). In this paper we only focus on OpenMP programming: we thus consider scalar IEEE (i.e. not instrumented with CADNA) codes and scalar CADNA codes in order to precisely measure the performance impact of CADNA for OpenMP multi-threaded applications.

4.1 A reduction code

This first experiment shows that a sequential code and its parallel version may exhibit a very different numerical quality.

The sum S of all elements of an array A of size $2n$ with $n = 10^6$ is computed in single precision. For $i = 0, ..., 2n-1$, each element $A[i]$ is initialised as $A[i] = -i$ if i is even, and $A[i] = i$ otherwise. Therefore the exact sum is 10^6. The sum S is computed using an OpenMP `for` loop construct with a `reduction` clause and two possible scheduling clauses: `(static)` so as the loop is divided into chunks of size $2n$ over the number of threads, and `(static,1)` so as the chunk size is 1. Results obtained for 1, 32, and 240 threads, with and without CADNA, are reported in Table 1. As already mentioned in Sect. 2.2, CADNA prints only the exact significant digits of results and @.0 if they are numerical noise.

Using the `(static)` scheduling option, whatever the number of threads, the result S is 1.000000E+06 with or without CADNA. Therefore the result has the maximum possible accuracy in single precision, i.e. 7 exact significant digits. Indeed, in this case each thread has in charge contiguous elements of A and alternatively sums positive and negative values: all computed results are accurate.

Using the `(static,1)` scheduling option, with 32 or 240 threads, the result has no correct digits and is provided by CADNA as @.0. Without CADNA the result depends on the number of threads and one can notice that for 240 threads

[6] http://ispc.github.io

	`(static)`		`(static,1)`	
# threads	without CADNA	with CADNA	without CADNA	with CADNA
1	1.000000E+06	1.000000E+06	1.000000E+06	1.000000E+06
32	1.000000E+06	1.000000E+06	1.892352E+06	@.0
240	1.000000E+06	1.000000E+06	1.617920E+05	@.0

Table 1. Results S obtained with and without CADNA using two possible schedulings.

it has a wrong order of magnitude. Each thread sums indeed values of the same sign, the absolute value of the sum thus keeps on increasing, and its accuracy regularly decreases due to the limited floating-point precision. The reduction involves inaccurate results with close absolute values and different signs and thus provides numerical noise. In this case, CADNA detects cancellations, i.e. losses of accuracy due to the subtraction of close inaccurate results.

One advantage of this simple numerical example is that its exact result can be easily compared to the computed one. It points out that the accuracy of results may change from a sequential to a parallel execution of a code and also among several parallel executions with different schedulings. This numerical example emphasizes the need for a CADNA version available for OpenMP codes.

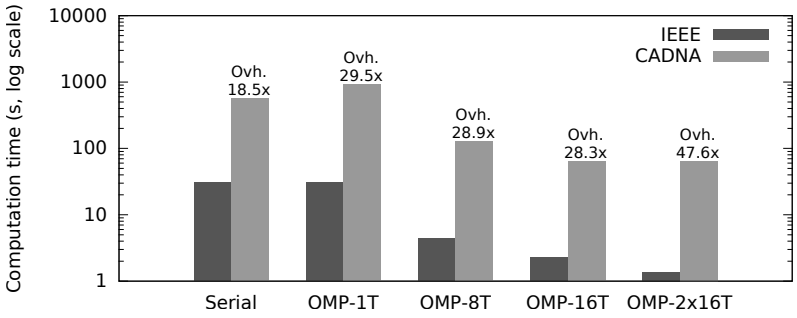
4.2 Performance tests

We now focus on the performance impact of CADNA on OpenMP codes by studying different micro-applications or benchmarks. In order to handle NUMA effects on *CPU-server*, we use the following OpenMP settings for thread affinity policies:

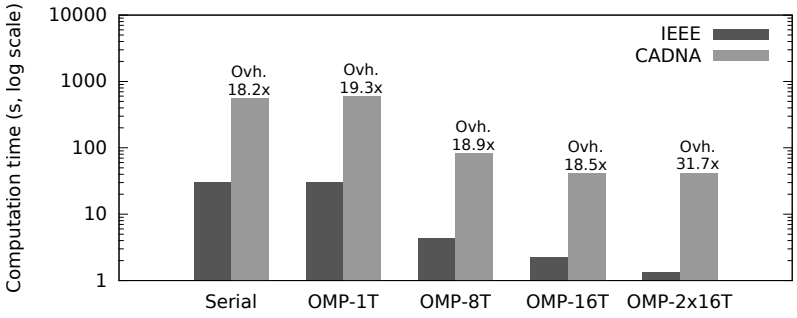
- when the number of threads is lower or equal to 8, `OMP_PLACES` is set to `sockets(8)` and `OMP_PROC_BIND` to `master` in order to have all threads on the same socket as the master thread;
- otherwise (i.e. for 16 or 2×16 threads) no thread affinity policy is used (`OMP_PROC_BIND` set to `false`): we let the OS scheduler run the threads on all physical CPU cores (with 16 threads, i.e. without SMT) or on all logical CPU cores (with 2×16 threads, i.e. with 2-way SMT).

On *Xeon-Phi*, no thread affinity policy is used. By default, we use `gcc/g++` version 4.9.2, and Intel `icc/icpc` version 2017 (beta).

Figures 1 and 2 first present performance results for a Mandelbrot set computation (4096×4096 pixels, with a maximum number of 4096 iterations per pixel) and for a 3D finite difference (FD) stencil (6 iterations over a $256 \times 256 \times 256$ grid with a six-order central stencil). First, for serial runs the CADNA over IEEE overheads match the ones previously obtained [4]. When considering OpenMP runs with only one thread, one can see an important additional overhead with gcc (see Figs. 1(a) and 2(a)). This seems to be related to the handling of `threadprivate` variables by gcc. Such a problem does not exist with icc (see Figs. 1(b) and 2(b)).



(a) With gcc



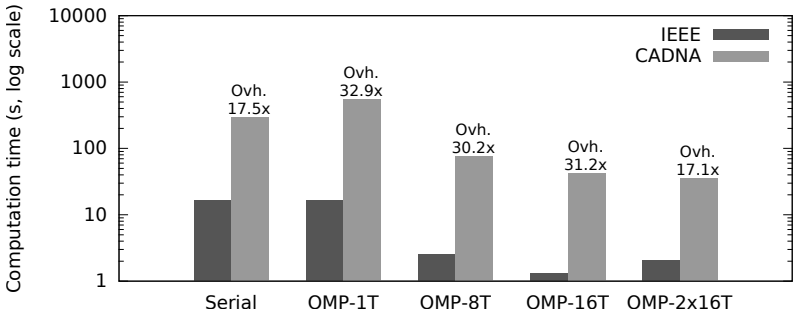
(b) With icc

Fig. 1. Performance results for Mandelbrot computations, along with CADNA over IEEE overheads (`Ovh.`). `OMP-XT` denotes OpenMP runs with `X` threads.

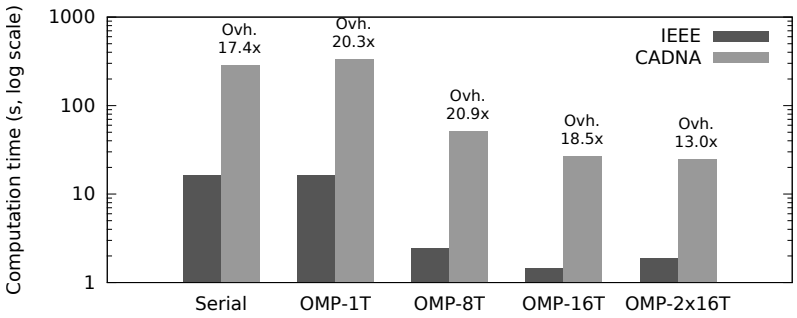
As far as Mandelbrot computations are concerned (see Figs. 1(a) and 1(b)), the CADNA overheads remain stable when the number of physical CPUs used increases. This shows no scaling overhead with CADNA for compute bound applications. One can however notice that, contrary to the IEEE version, the CADNA version shows no performance gain with the SMT capability.

As far as 3D FD stencil computations are concerned (see Figs. 2(a) and 2(b)), the CADNA overheads remain stable (or decrease somewhat) when the number of physical CPUs used increases. This also shows no scaling overhead for CADNA for such an application which is memory-bound. Here again the CADNA version shows no performance gain with the SMT capability, but this is also the case for the IEEE version since this application is memory-bound.

Figure 3 presents results for the `+` reduction. The decrease in CADNA overheads between the serial execution and the OpenMP with 1 thread is only due to a slower computation for the IEEE code with 1 OpenMP thread compared to the serial IEEE code without OpenMP. When using multiple threads, one can see that the CADNA overheads are lower. This is due to the fact that the arithmetic intensity of the benchmarks is modified by the use of CADNA. Using



(a) With gcc



(b) With icc

Fig. 2. Performance results for FD stencil computations, along with CADNA over IEEE overheads (`Ovh.`). `OMP-XT` denotes OpenMP runs with `X` threads.

CADNA, we have indeed at least 3 times more computations (for each sample, and for the extra operations associated with the rounding mode emulation and the random number generation), while requiring 4 memory accesses. But these memory accesses can be performed at once with the same cache line (as the members of the stochastic types are contiguous in memory): the arithmetic intensity thus increases with CADNA. Since this reduction benchmark presents an even lower arithmetic intensity than the 3D FD stencil one, its performance is even more limited by the memory bandwidth. Using CADNA thus leads to performance less limited by the memory bandwidth and to greater speedups when increasing the number of threads and when using SMT. Similar conclusions have been obtained for the `*` reduction.

In order to study the impact of `atomic` constructs required by the instability detection, we present in Fig. 4 performance results for the Mandelbrot set computation with detection of no instability and of all instabilities. The increase of CADNA overheads between serial and 1-thread OpenMP executions is thus 4.5% for no instability (hence no `atomic` construct) in Fig. 4(a) and 6% for self-validation only (some `atomic` constructs used) in Fig. 1(b): the `atomic`

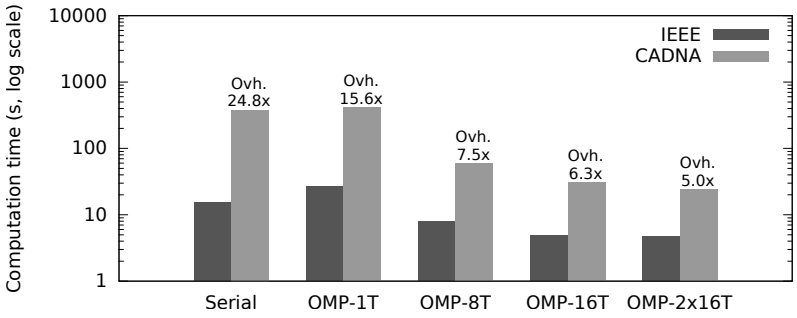


Fig. 3. Performance results (using icc) for reductions (`+` operator on 2^{28} elements), with CADNA over IEEE overheads (`Ovh.`). `OMP-XT` denotes OpenMP runs with `X` threads.

construct impact is thus not significant for 1 OpenMP thread. When detecting all instabilities (all `atomic` constructs used) in Fig. 4(b), this increase is even lower (2.1%) since the CADNA overheads are much more important. Moreover, the `atomic` construct impact does not increase for multiple OpenMP threads as shown in Figs. 1(b), 2(b), 4(a) and 4(b).

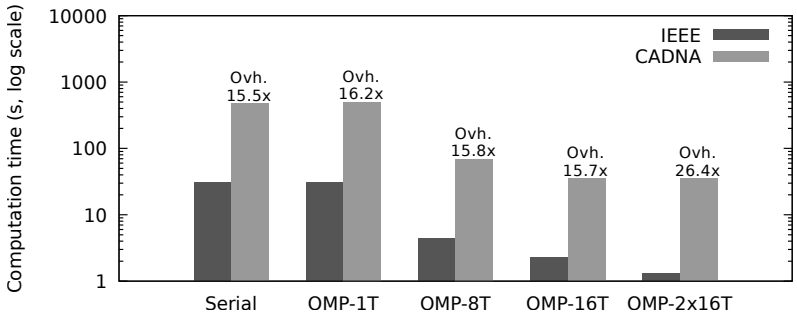
Finally, Fig. 5 presents the CADNA over IEEE overheads of our benchmarks for serial and OpenMP executions on *Xeon-Phi*. The CADNA overheads are greater on this architecture for Mandelbrot and reduction computations due to the use of the `strict` and `no-except` floating-point models (required by CADNA with `icc`): contrary to *CPU-server*, such models imply an important overhead on the Knight Corner architecture (e.g. 5.1x for Mandelbrot computations). Nevertheless these tests confirm the previous conclusions (obtained on *CPU-server*), here for a higher number of cores and threads: no scaling overhead for CADNA (and even lower overheads for memory-bound applications with SMT), and no extra overhead due to the `atomic` constructs used for the instability detection.

The overhead associated with the `atomic` constructs on stochastic variables will be studied in the next section.

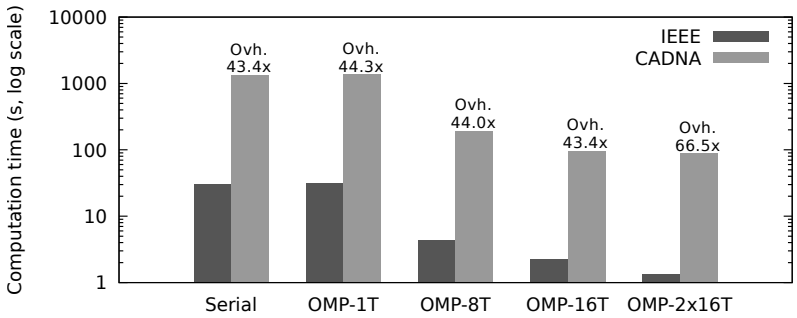
4.3 A shallow-water application

Presentation. The CADNA library is finally applied to a "real-life" application: a Shallow-Water model (denoted SW). A SW model is a numerical integration of the SW equations [18] that describe the flow in a fluid. The application presented here describes the evolution of the water height and velocities in a two-dimensional oceanic basin (the water velocity is constant on the water depth). Fundamentally, a SW computation can be viewed as a combination of finite difference stencils.

In our application, the model is integrated for 500 time steps on a 256×256 grid. There are two integration modes: the direct model (denoted *forward*) that



(a) No instability detection



(b) Detection of all instabilities

Fig. 4. Performance results for Mandelbrot computations (using icc), along with CADNA over IEEE overheads (`Ovh.`). `OMP-XT` denotes OpenMP runs with `X` threads.

computes the output of the model, and the adjoint model (denoted *backward*) that computes the sensitivity of the output to the initial conditions. Both modes (forward and backward) involve the same set of computations, the only difference lies in the direction of the time loop: from past to future in the forward mode, from future to past in the backward mode. As a consequence, the forward mode implementation is fully parallel in space using a space domain decomposition, whereas the backward mode requires the use of numerous `atomic` operations.

The successive computations of the forward and backward modes allow the estimation of a scalar quantity called the "residual" which is a key parameter for validating adjoint codes [3]. The code was compiled using `icc/icpc` version 2015, and run on *CPU-server*.

Performance and numerical results. We first present the performance impact of CADNA on this SW code. Figure 6 presents performance results of the forward mode. It shows that there is no scaling overhead in CADNA. The overhead is even lower with multiple threads (9.9x for 16 threads) than for one

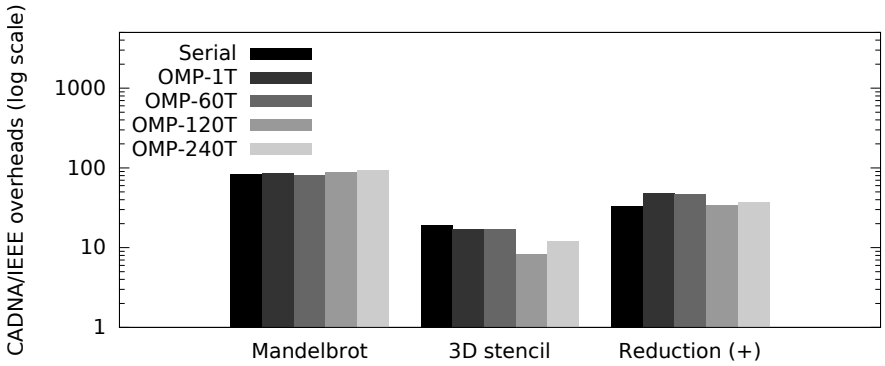


Fig. 5. CADNA over IEEE overheads for various benchmarks on the Xeon Phi, using icc and the same parameters than on *CPU-server* (except for the reduction here with 2^{27} elements due to memory constraints).

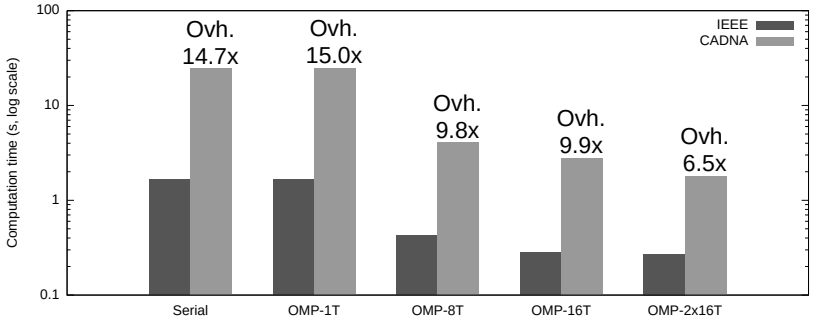


Fig. 6. Performance results (using icc) for Shallow-Water (forward mode), with CADNA over IEEE overheads (`Ovh.`). `OMP-XT` denotes OpenMP runs with `X` threads.

thread (15.0x) as it was already explained in Sect. 4.2 for such memory-bound application.

Figure 7 presents performance results of the backward mode. In this case, parallel speedups are limited by the use of numerous `atomic` operations on floating point numbers, as already discussed in [13]. With CADNA, these operations must be converted into `critical` regions as shown in Sect. 3 (in this SW model, we use three distinct `critical` regions). As a consequence, the computation time and therefore the overhead are increasing with the number of threads, because of increased lock contention. It should thus be noticed that the numerical validation of a parallel code using (numerous) `atomic` directives remains possible using CADNA but with a higher cost.

Finally we focus on the numerical stability of the residual of the SW model (see Table 2). For clarity only the serial and 16-thread OpenMP results are presented. It can be noticed that there are only 6 common digits between the

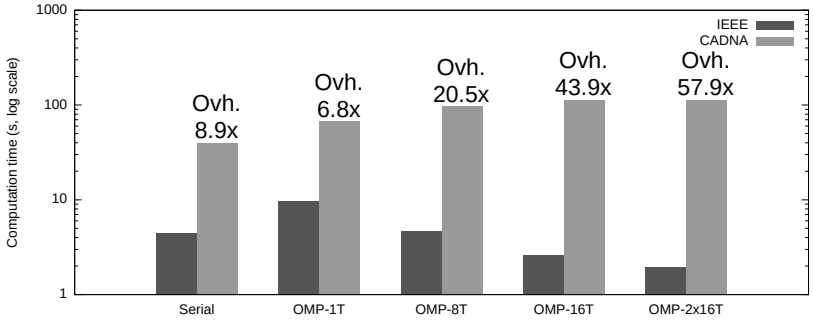


Fig. 7. Performance results (using icc) for Shallow-Water (backward mode), with CADNA over IEEE overheads (`Ovh.`). `OMP-XT` denotes OpenMP runs with `X` threads.

	IEEE	CADNA
Serial	3.446611873236805E-06	3.4461E-06
OpenMP - 16 treads	3.446619149194419E-06	3.446E-06

Table 2. Comparison between IEEE and CADNA computations of the Shallow-Water residual for serial and parallel executions.

results computed without CADNA by the serial and parallel versions. CADNA estimates that 5 digits in serial and 4 digits in parallel are correct. These digits (except the last one in serial which may not be reliable in CADNA) are consistent with the IEEE versions. Thanks to CADNA, we can therefore state that the OpenMP version of this SW application is reliable with a 4-digit accuracy. Moreover, by comparing the serial and parallel numerical results, we can deduce that the differences in the IEEE results are likely due to round-off error propagations and not to a bug in the OpenMP parallelization.

5 Conclusion

In this paper, we have presented a new OpenMP-enabled CADNA version that can be used to validate parallel codes by providing the number of exact significant digits of their results, and that presents similar or lower CADNA overheads with respect to the serial ones. This library can also help to explain differences in the numerical results between serial and (multiple) parallel executions.

It is worth noting that the control of accuracy can also be performed on MPI codes [12]. Therefore the work presented on this paper could be extended to enable the numerical validation of large scale applications using both MPI and OpenMP. Besides, the CADNA instrumentation of codes could be eased thanks to OpenMP-like compiler directives that would avoid the insertion and modifications of C/C++ instructions in the user program. A preliminary study has been performed on such directives. However work has still to be carried out

to propose a complete set of directives that enable all CADNA functionalities and can control the numerical quality of any C/C++ program.

6 Acknowledgments

The authors thank Pierre-Emmanuel Le Roux (LIP6) for managing the compute servers, and Philippe Thierry (Intel) for providing the Intel compiler (2017-beta). They also thank the reviewers for their helpful comments.

References

1. G. Alefeld and J. Herzberger. *Introduction to interval analysis*. Academic Press, 1983.
2. OpenMP Architecture Review Board. OpenMP Application Program Interface, Version 4.0, July 2013.
3. J. Brajard, P. Li, F. Jézéquel, H.-S. Benavidès, and S. Thiria. Numerical Validation of Data Assimilation Codes Generated by the YAO Software. In *SIAM Annual Meeting, San Diego, California (USA)*, July 2013.
4. P. Eberhart, J. Brajard, P. Fortin, and F. Jézéquel. High performance numerical validation using stochastic arithmetic. *Reliable Computing*, 21:35–52, 2015.
5. C. A. Hölbig, A. Do Carmo, and L. P. Arendt. High accuracy and interval arithmetic on multicore processors. *Computational and Applied Mathematics*, 32(3):425–434, 2013.
6. IEEE and The Open Group. The Open Group Base Specifications, 7th edition, 2013. `http://pubs.opengroup.org/onlinepubs/9699919799`.
7. Intel. User and Reference Guide for the Intel C++ Compiler 15.0. https://software.intel.com/en-us/compiler_15.0_ug_c, 2015.
8. F. Jézéquel and J.-M. Chesneaux. CADNA: a library for estimating round-off error propagation. *Computer Physics Communications*, 178(12):933–955, 2008.
9. F. Jézéquel, J.-L. Lamotte, and I. Said. Estimation of numerical reproducibility on CPU and GPU. In *Proc. of the 2015 Federated Conference on Computer Science and Information Systems*, volume 5 of *Annals of Computer Science and Information Systems*, pages 675–680. IEEE, 2015.
10. U.W. Kulisch. *Advanced Arithmetic for the Digital Computer*. Springer-Verlag, Wien, 2002.
11. C.R. Milani, M. Kolberg, and L.G. Fernandes. Solving dense interval linear systems with verified computing on multicore architectures. In *High Performance Computing for Computational Science - VECPAR 2010 - 9th International conference, Berkeley, CA, USA, June 22-25, 2010, Revised Selected Papers*, volume 6449, pages 435–448. Springer, 2010.
12. S. Montan and C. Denis. Numerical verification of industrial numerical codes. In *ESAIM: Proc.*, volume 35, pages 107–113, March 2012.
13. L. Nardi, F. Badran, P. Fortin, and S. Thiria. YAO: a generator of parallel code for variational data assimilation applications. In *Int. Conf. on High Performance Computing and Communication*, pages 224–232. IEEE, 2012.
14. N. Revol and P. Théveny. Parallel implementation of interval matrix multiplication. *Reliable Computing*, 19(1):91–106, 2013.

15. J. Vignes. Zéro mathématique et zéro informatique. *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, 303:997–1000, 1986.
16. J. Vignes. A stochastic arithmetic for reliable scientific computation. *Mathematics and Computers in Simulation*, 35(3):233–261, 1993.
17. J. Vignes. Discrete Stochastic Arithmetic for validating results of numerical software. *Numerical Algorithms*, 37(1–4):377–390, December 2004.
18. C.B. Vreugdenhil. *Numerical Methods for Shallow-Water Flow*. Springer Netherlands, 1994.
19. M. Zimmer. Using C-XSC in a multi-threaded environment. Technical Report BUGHW— WRSWT 2011/2, Universität Wuppertal, Germany, 2011.